AERCO RP/M Release 1.1

CP/M programs typically require a number of keys not defined on the TS-2068 keyboard. They are implemented as follows:

CONTROL KEY: This is a prefix key used much like the caps shift or symbol shift keys. Full scale keyboards have a dedicated control key that is held depressed while some other key is pressed. In our case, the control key prefix is generated with caps shift and symbol shift as the E cursor is normally generated. If a program requires a Control Z for example, hold caps shift while pressing symbol shift, release both keys, and press the Z key.

CONTROL C: This is a very commonly used command that generally means "exit" from a program or a process. It of course can be generated as described above, but more conveniently as Caps Shift 1

ESCAPE KEY: This is a prefix key that is normally expected to be pressed before a specified key. For example, if a program needs an Escape Z, you would press escape and release it, then press Z.
In our case ESCAPE is Caps Shift 9; so in the above example, you would hold Caps-Shift depressed while pressing 9, then release Caps-Shift and press Z.

BACKSPACE: Caps Shift 0; as usual.

COMMONLY REQUIRED COMMANDS:

1. To freeze output on the screen: **CONTROL S** Example... use the built-in command **TYPE** to view a text file on the screen. To freeze the display for detailed reading, type **CONTROL S**. Any key will re-start the display scrolling. Be sure to enter TYPE FILES.DOC

2. To send screen output to the AERCO Centronics printer interface, type **CONTROL P**. Screen output will now also be sent to the printer. **CONTROL P** again will toggle the printer state, **CONTROL C** will turn the printer off (and halt any on-going process).

AERCO

DO THIS FIRST

MAKING A COPY OF THE RP/M BOOT DISC:

The distribution disk should always remain write-protected and be used only to make copies of all or some of the programs on it. The following instructions assume you have at least 2 drives.

1. **BOOT the System.** Place the distribution diskette in drive A and turn the computer on. Within a few seconds the RP/M prompt **a0>** should appear. (The **a** means that drive A is currently selected, the **0** means that user 0 is currently active, and the **>** is the standard CP/M prompt character.) You can view the directory on the disk if you wish by pressing **D** and **ENTER**. (You loaded and executed the disk file D.COM)

2. **FORMAT a blank Diskette:** Place a blank diskette in drive B. Type **FORMAT68** and press **ENTER.** (You loaded and executed FORMAT68.COM) This program will ask which disk you wish to format; reply **B**. It will then warn you that there is data on the disk and ask your permission to erase it. Reply **Y**... we hope that disk didn't have the only copy of your last three years tax records! The program will print a * for each track written in the inimitable AERCO style and then ask if you want to install the system tracks on the disk. Reply **Y.** Four more * for the system tracks and we're back to the beginning. If you have a bunch of diskettes you want to initialize, now is a good time to do it. Simply pressing **ENTER** will start the process again on drive B. When you are finished formatting, type **CAPS-SHIFT 1** on the TS-2068 keyboard or **CONTROL C** if you are using a full-size terminal. (CONTROL C is the standard CP/M command to exit a program or halt a process. It tells the system to do a **WARM BOOT** which terminates your program and brings you back into the command mode with a system prompt.)

3. **COPY the Files:** Type **RPMPIP B:=A:*.*** and press **ENTER.** The system will commence to copy all files from drive A to drive B. If you wanted to copy only the COM files, you would specify ***.COM** instead of ***.*** . If you wanted to copy all extensions of the file POOP, you would type **POOP.*** instead of ***.*** . (* is called a **WILDCARD** character. It signifies "EVERYTHING". Another useful wildcard character is **?**. It signifies "ANY ONE CHARACTER".) Note that PIP (Peripheral-Interchange-Program) is a VERY powerful general purpose utility that has many pages of options and features. It is a good candidate for careful study. We will be providing more and better documentation as time goes by, but it will never be a total substitute for a good CP/M textbook. Any large bookstore should have several CP/M nanuals... pick one that seems appropriate and have fun!

4. **DON'T Write Protect the disk:** Many CP/M programs expect to be able to write on the disk for the automatic creation of Backup files and many other purposes. (CP/M programs will report the inscrutable error message **BAD SECTOR** no matter what they don't like about the disk.) **ALWAYS** make a work copy of important programs and back-up your work as appropriate. It's a very wise policy to open the door on your drives when the system power is turned on or off.

PREFACE

Under the assumption that the reader is an assembly language programmer familiar with ASM and DDT, the material in this manual is presented at a somewhat advanced level. For others less experienced who would benefit from an introduction to these topics, a good place to find such an introduction would be in "CP/M Revealed," published by Hayden (order phone 1-800-631-0856 or 201-843-0550 in NJ) and available at all Dalton bookstores.

Programs that run under CP/M 2.2 will run without modification under RP/M. However, console and disk processing procedures under RP/M are similar to CP/M; they are not identical. RP/M assumes a video display console. RP/M file open functions search simultaneously in user area zero and in the current user area. File delete, rename, and close functions, search only in the current user area. After bringing up RP/M, go to each nonzero user area containing active files and rename there any file having the same name as an active file in user area zero. Common access to files in user area zero is a major improvement; techniques for accomodating the minor disadvantage alluded to here are discussed further at the end of Chapter 1.

Chapter 2 provides information on disk file structure and on my unique nomenclature which you will find necessary for understanding the disk file processing routines in RDOS. Two utility programs designed to help cope with certain types of disk problems are discussed in Chapter 3.

Jack D. Dennon
November 8, 1983

CONTENTS

Introduction to RP/M

RP/M is functionally equivalent to CP/M 2.2. Under RP/M, the resident of CP/M version 2.2 is replaced by a new resident console processor called RCP, and a new resident disk operating system called RDOS. These modules replace the console command processor, CCP, and the basic disk operating system, BDOS, of CP/M version 2.2. Advantages of running under RP/M include advanced operational features such as a paged TYPE display, file size indication in the DIR display, cross drive file search, and access to user area zero from all user areas.

Users familiar with CP/M will feel at home with RP/M. All the familiar built-in commands are present, and almost all operate as expected. An exception is the file erase command, the syntax of which has been changed both to tighten the code and reduce the chance of unintentional erase.

RCP Built-in Commands

DIR d:afn

The entire argument is optional; if omitted, the command is the same as DIR *.*, names of all files on the current disk will be displayed. The size of each file is indicated by displaying information obtained from the record count byte, and also the extent byte, of the zeroth physical extent. In the example DIR display

```
A: DBSORT   HLP  22 : DBQ      OVL + 4 : DBCOPY   TBL   1
```

the file DBSORT.HLP is shown to occupy 22 pages of disk space. A page is 256 bytes. The file DBQ.OVL is a large file occupying more than one logical extent. In this case, the page count of 4 is not too significant; all we know is that DBQ.OVL is a large file. The plus sign (+) will occur only for disks arranged with more than one logical extent per physical extent. On standard 8" SSSD floppy disks, where one logical always occupies one physical extent, the size of a large multi-extent file, no matter how large, will always appear in the DIR display as 64 pages. The DIR displayed page size is quantitative for small files, but is qualitative for large files.

ERA d:afn

The disk name d must be explicitly stated. An ambiguous filename will be accepted; all files matching the ambiguous prototype will be erased. For example,

```
ERA  B:*.BAK
```

will erase on drive B all files of type BAK. The "erase all" command must state the request twice, as in the example

```
ERA  C:*.*  C:*.*
```

which will erase all files on the disk in drive C. The general form of the command to erase all files is

```
ERA  x:*.*  d:*.*
```

The drive name x is ignored and may be omitted. The drive name d must be explicitly stated.

TYPE d:ufn

The drive name d is optional; the filename must be unambiguous. Scrolling of the display will pause after 24 lines. Any keystroke, except control-C or control-P, will advance page. Control-P will advance page and toggle the list flag. Control-C will warmboot. Page size may be changed with the PAGE command.

PAGE n

Page size for the TYPE command will be set to the decimal value n which may be in the range 0 through 255. Page size zero disables paging. With paging enabled or disabled, scrolling can be paused with control-S.

SAVE n d:ufn

The memory image starting at location 0100H and extending for n pages of 256 bytes each will be written to the disk in drive d as the file named ufn. The page count may be stated in decimal or in hexadecimal. For example, the two commands

```
SAVE  25  B:PROG.COM
SAVE  H:19 B:PROG.COM
```

are effectively identical. The drive name is optional; the filename must be unambiguous.

REN a:new=b:old

The drive name a and drive name b are optional; if stated, they must be the same. The filenames new and old must be unambiguous and the new name must be a filename not already present in the directory.

USER n

The user number n should be stated in decimal and be in the range 0 through 15. When operating in a nonzero user area, the system prompt will display the active user.

Multiple commands on a single line

Multiple built-in commands can be stated in a single console command line.

User areas under RP/M

File access across nonzero user area boundaries is not permitted, however, all users may access the files assigned to user area 0. This added capability brings with it an added programmer responsibility. It is the programmer's responsibility to avoid creating in any nonzero user area, a file having the same name as a file existing in user area zero. This should in practice present no difficulty since the DIR command always displays simultaneously the files in user area zero and the files in the current user area, and identifies those belonging to the current user.

User area disipline

For those making use of the user area facility under RP/M it would be good practice to dedicate user area zero exclusively to those files needed in all user areas, and in user area zero permit creation of no temporary files whatever. The useful advantage of access to user area zero from all user areas is obtained by permitting the file OPEN operation to examine simultaneously the directory entries belonging to user area zero and those belonging to the current user. However, the file CLOSE operation is permitted to examine only those directory entries belonging to the current user. To complete the file security measures that this arrangement supports, if you are going to use nonzero user areas, then a good practice would be to dedicate user area zero exclusively to permanent files marked read-only, and do all of your work in nonzero user areas.

On many systems it will, of course, make sense to simply ignore the user area facility and do all work in user area zero. The point is, under RP/M, it is inadvisable to dabble with the user area facility. Either ignore the facility entirely, or adopt a disipline that will avoid creating in user area zero a file named the same as a file in a nonzero user area.

Disk File Processing

RP/M implements the CP/M disk data structure for file processing. CP/M divides every disk into two distinct areas, a system area that is ignored by BDOS, and a data area that is processed by BDOS. The system area originally was used by CP/M 1.3 to reserve a few tracks for operations concerned with booting the system, however, the quantity called system track "offset" has been generalized under CP/M 2.2 to partition a large physical disk, that is, a disk holding more than 8 megabytes, into two or more smaller logical disks each holding 8 megabytes or less.

The restriction to 8 megabytes arises from two CP/M parameters. The disk entity that is addressed is a "record" containing 128 bytes, and, the disk address of each such record, called the disk record ordinal, is a 16-bit quantity. On the basis of these parameters the BDOS of CP/M 2.2 can address up to 8,388,608 bytes, abbreviated 8 megabytes, or 8 Mb. This limitation pertains to the data space of the logical disk as processed by BDOS.

The offset used to mark the physical starting track of a logical disk device is itself a 16-bit value. This parameter coupled with the ability of CP/M 2.2 to process up to 16 logical disk devices permits CP/M to manage either one large disk or several smaller disks containing in the aggragate up to 128 megabytes of disk storage space.

Disk Reservation

For disk reservation purposes, the 128 byte records in the data area of a logical disk are grouped into blocks of 8, 16, 32, 64, or 128 records, called record reservation blocks, or reservation blocks, or record blocks, or sometimes just blocks. For disk addressing purposes, each record block must be given a unique name or number. When the disk data space is divided into fewer than 256 blocks, the block name can be an 8-bit quantity called a record block reservation byte. When the data space is divided into more than 255 reservation blocks, the block name will be a 16-bit quantity called a record block reservation word.

The data area of a logical disk is divided for BDOS--at a block boundary-- into a disk directory area and a disk file area. The directory area always begins with block 0 (zero) and may contain from one through sixteen contiguous blocks. Each entry in the disk directory occupies 32 bytes of storage space. The first half of each entry contains a logical file name; the second half of each directory entry contains the names of the reservation blocks assigned to the file.

File control block (fcb) format.

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
FNT	00	dr	f1	f2	f3	f4	f5	f6	f7	f8	t1	t2	t3	ex	00	s2	rc
RBT	10	b0	b1	b2	b3	b4	b5	b6	b7	b8	b9	bA	bB	bC	bD	bE	bF
	20	cr	r0	r1	ov												

dr	fcb drive code	00 = no selection, use current drive 01 = drive A 02 = drive B, etc.
dr	dir	E5 = erased entry 00 = file assigned to user 0 0n = file assigned to user n
f1-f8 fx'	file name high order bit of each byte is an "attribute" flag f1'-f4' (spare) f5'-f8' (reserved)	
t1-t3	file type t1' set = file R/O t2' set = system file t3' set = file has been archived	
ex	low five bits of the logical extent	
s2	low four bits hold high four bits of the 9-bit logical extent s2' set = file not written	
rc	record count of the current logical extent; takes on values 00 through 80 hex	
b0-bF	16 RBT bytes, or 8 RBT words, containing names of disk blocks occupied by the file	
cr	current record, sometimes called "next record," indicates next record to be accessed in the current logical extent	
r0,r1 ov	record ordinal for random access functions overflow from r1, should always be 00	

Fig. 2.1. Disk directory entry and file control block structures.

Record Block Table (RBT)

The first half of each directory entry is said to be an entry in the file name table, or FNT. The second half of each directory entry is said to be an entry in the record block table, or RBT. The disk directory may be viewed as the interlaced merger of these two tables whereby symbolic file names in the FNT are mapped onto actual disk locations by the record block names in the RBT.

Open File

To "open" a file means to copy a directory entry for a named file from the disk directory into a "file control block" area, or fcb, located in main memory at a place designated by the calling user program. As illustrated in Fig. 2.1, the fcb is essentially an image of a directory entry, with either one byte, or four bytes, appended. For sequential access operations, a single appended byte to hold the "current record" pointer cr is sufficient. For random access operations, four appended bytes provide space for the current record pointer, a 16-bit file record ordinal, plus one additional byte reserved for compatibility with future systems that may use a 24-bit record ordinal. Under CP/M 2.2, the additional fcb byte is called ov and is used only by BDOS function 35, compute file size, to return the high order bit of the hex value 10000 or 65,536, expressing the largest CP/M 2.2 record count possible.

File Name Table

Space is provided in the FNT for a primary file name containing from 1 to 8 characters occupying fcb bytes 01 through 08, and a secondary file name, called the file "type," containing from 0 to 3 characters occupying fcb bytes 09 through 0B. The primary and secondary file names are left adjusted in their respective fields with space (Ascii 20) fill.

File Extents

A large file needing more RBT space than is available in a single directory entry will use additional directory entries to hold the extended RBT. Additional directory entries for a large file are numbered for order identification by the contents of the "physical extent" bits that are contained in fcb bytes 0C and 0E, called "ex" and "s2," respectively. The low four bits of s2 together with the low five bits of ex form a 9-bit "logical extent" number as illustrated in Fig. 2.2. The "extent mask" divides the logical extent into a

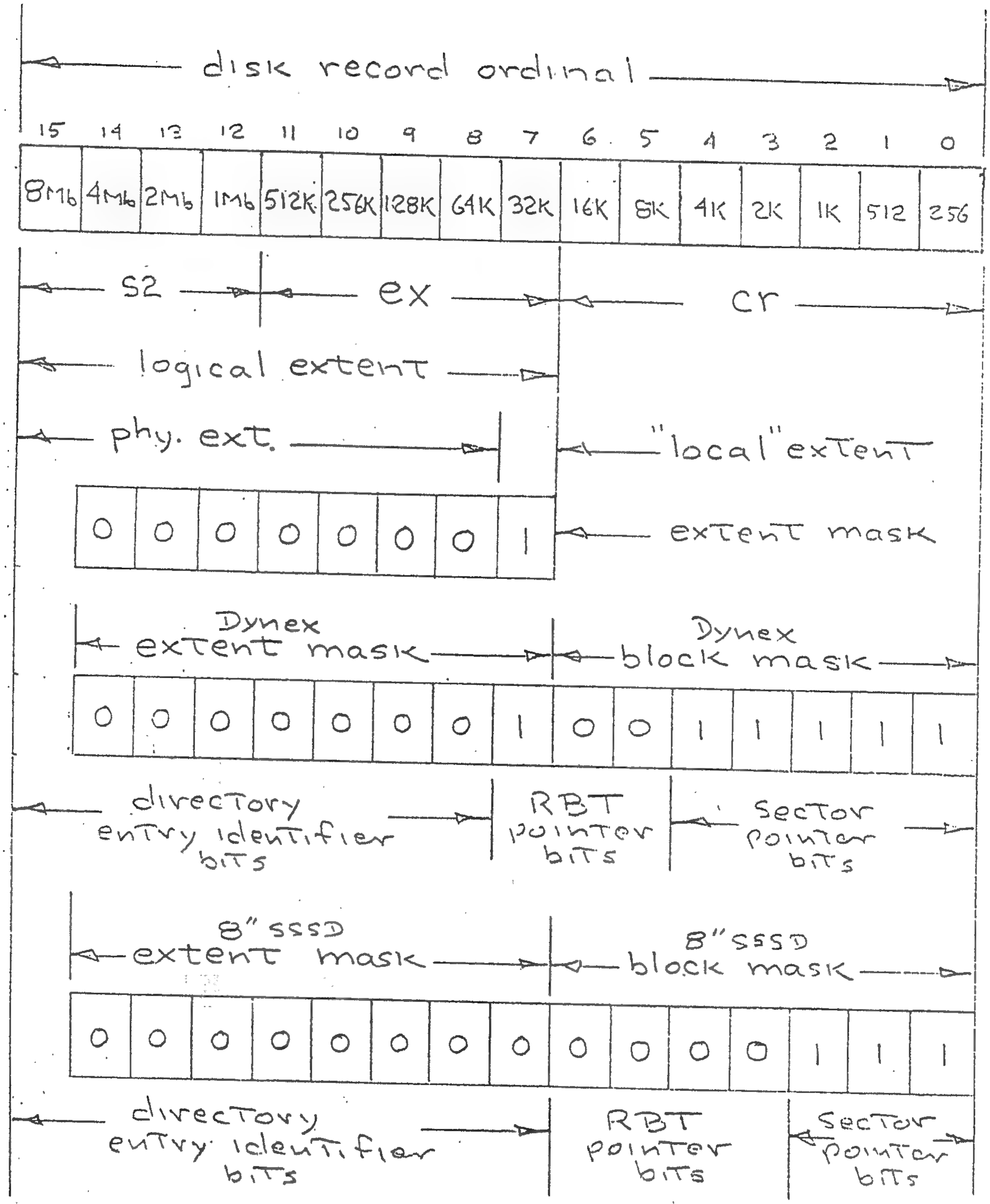

Fig. 2.2. Construction of a disk address from fcb values.

physical extent that numbers the order identification of the directory entry, and a "local extent" that identifies within the current directory entry a group of RBT entries within which the current record byte cr in turn singles out a specific record. This function of the extent mask is best illustrated by example. Fig. 2.2 includes two specific examples.

A large disk

The Dynex disk is arranged with 32 records in a reservation block. The block size is therefore 4k. There are 1215 of these blocks available, so 16-bit record block reservation words must be used to name blocks in the RBT. Each directory entry, or physical extent, can hold eight of these RBT words, so each physical extent can address up to 32k of disk space, or two logical extents. A logical extent always contains 16k. The local extent bit, in the Dynex case, identifies the logical extent that is being addressed, by pointing to one of the two logical extents contained in the current RBT. Within the current 16k logical extent, the contents of fcb byte cr always identifies a specific current record, numbered 0 through 127 decimal, or 0 through 7F hex.

A small disk

The standard 8" SSSD floppy disk is arranged with 8 records in a reservation block. The block size is therefore 1k. There are 243 of these blocks available, so record block reservation bytes can be used to name blocks in the RBT. Each directory entry, or physical extent, can hold sixteen of these RBT bytes, so each physical extent will address 16k of disk space, or one logical extent. The logical extent and the physical extent are identical on this disk, and so there are no local extent bits.

Block Mask

As illustrated by the two examples in Fig. 2.2, the 16-bit indirect disk address formed from s2, ex, and cr, is divided by the extent mask and the "block mask," into three fields that identify the directory entry order, the RBT byte or word within the directory entry, and finally the sector or record within the named record block.

Record block reservation (RBR) table

To expedite the search for unused record blocks, an additional table is constructed by BDOS in memory reserved for this purpose by CBIOS. In this table each record block is represented by a single bit. While the block is being used by a file, the bit is set (1), otherwise the bit is clear (0). This record block reservation (RBR) table is constructed by scanning all values in the disk directory RBT. Such construction of a disk RBR is called disk "log-in."

When new files are created, available blocks are found by scanning the RBR; their names are entered into the RBT in an fcb in memory, and the blocks are temporarily reserved by setting their reservation bits in the RBR. Only when a file is "closed," is the RBT image in the fcb copied to the actual RBT in the directory on the disk, permanently reserving the named blocks.

Physical Disk Addresses

The track T that contains the first sector of the reservation block B is given by

$$T = \text{offset} + \text{record ordinal} \ / \ \text{Spt}$$

where

record ordinal = B x Blksize
Blksize = 8, 16, 32, 64, or 128
        = block size in sectors, corresponding
          to a "block shift" of 3, 4, 5, 6, or 7
Spt = sectors per track
offset = number of system tracks

For any disk available to the system, the parameters offset, block shift, and sectors per track, can be obtained from a disk parameter table in CBIOS. Under RP/M, the record ordinal is calculated in RDOS subroutine CRO; the track is calculated in RDOS subroutine PPD.

Creating a Disk Flaw Table

An unuseable spot on a disk can be effectively removed by permanently reserving the record block containing the unuseable sector. Since permanent reservation of referenced blocks is precisely what a directory entry provides, to create a disk "flaw table" we need only create a directory entry referencing the block that contains the bad sector.

The problem is, when we get a disk error we don't normally know the block number, and furthermore, we have no direct way to its discovery. In this chapter we introduce two programs designed to cope with this problem. FLAWTBL is a disk testing utility program that can discover bad spots on a disk by reading each sector in the disk data space. FLAWTBL will run under either CP/M or RP/M. LOCKOUT is designed specifically to take advantage of special flaw table features built into RP/M.

FLAWTBL

FLAWTBL "repairs" disks by reserving unuseable record blocks in a flaw table on the disk. FLAWTBL works through the RDOS of RP/M, or the BDOS of CP/M--rather than around it as do some disk test programs--by temporarily vectoring disk data error processing directly into FLAWTBL. All record blocks, except block 0, are tested via standard system logical record reads. Not tested by FLAWTBL are the "system" tracks, and logical record block 0. Under either RP/M or CP/M, logical record block 0 is always occupied by the disk directory and cannot be addressed by standard read requests. Unrecoverable errors on system tracks are of no concern on a data-only disk, while such errors in logical record block 0 render the disk unuseable under either RP/M or CP/M. FLAWTBL is designed to test disk data space. Faulty system or directory space cannot simply be mapped out of use.

FLAWTBL is called by a console command such as

```
FLAWTBL d:
```

where d is the name of the drive containing the disk to be tested.

FLAWTBL will perform standard logical record reads on all records of each reservation block numbered greater than 0. To interrupt testing, and return control to the operating system, press a keyboard key such as space, or carriage return.

FLAWTBL Console Messages

```
Number of flawed blocks = xxxx
```

The number xxxx is the count in hex of the blocks reserved.

```
SELECT ERROR FROM CBIOS.
```

FLAWTBL was unable to obtain disk parameters from CBIOS, under RP/M or CP/M 2.2. No flaw table will be written.

```
Flawed block = xxxx
```

Block xxxx (hex) had an error and will be reserved in the flaw table. May be accompanied by a diagnostic message from your CBIOS.

```
Good disk!
```

No disk errors were encountered.

Flaw Tables

Flaw table directory entries are marked "system" type under RP/M and under CP/M 2.2, and therefore will be invisible to the DIR directory list command. Flaw table directory entries can be displayed by

```
STAT d:*.*
```

or by

```
STAT d:*.TBL
```

where d is the disk drive name.

CBIOS Error Processing

In order for FLAWTBL to regain control following a disk data error, your CBIOS must simply set an error code and return. Any CBIOS that attempts to take matters into its own hands by, for example, waiting for a console character or control-C, may render FLAWTBL unuseable. The specification for CBIOS calls for it to set an error code and return, so normally you will not encounter such a problem.

Double Density Floppy Disks

It is important to understand that FLAWTBL is in no way hardware dependent. FLAWTBL reads your disk in exactly the same way that any standard user program will read your disk: By making logical record read requests to the operating system. What this means, for example, is that if a disk tests error free under FLAWTBL but still gets read errors the next time you try to use it, you should test the disk several times. Erratic results could indicate a drive alignment or positioner problem, contamination on the read head, or mishandled, damaged, or wornout media. Controller problems are possible, but less likely. With higher capacity non-IBM formats, reliability generally will vary inversely with length of the physical record. As more of the disk surface is dedicated to data, with less available for the formatting information upon which the controller depends for synchronization, the margin for error is diminished.

For example, on an 8" double density disk, Micromation puts 52 logical records on each track by blocking two logical records in each of 26 physical sectors, using the standard IBM System 34 double density format. On this same physical disk, Pickles & Trout and many other system integrators put 64 logical records on each track by blocking four logical records in each of 16 physical sectors. Other things being equal, the 64 record-per-track format will be more demanding of drive and media quality than will be the 52 record-per-track format. The standard IBM System 34 double density format provides 26 synchronizing record preambles on each track; while the higher capacity non-IBM format provides 16 such preambles per track. Reliability is inevitably proportional to redundancy. This needn't suggest that higher capacity formatting is a bad idea; it just means there is less margin for error and the quality of the drive and the media becomes more critical.

An Automatic Flaw Table Mechanism

A problem with the type of disk testing created by FLAWTBL, or any similar program, is that disk write errors are sometimes data dependent. A data dependent error may escape detection by the disk testing program, only to appear while running an important applications program.

What is needed is a way to trap disk errors immediately when they occur, and automatically enter the unuseable block into the disk flaw table. Although you can evade floppy disk write errors by discarding the disk, a flaw table type disk repair scheme is advantageous. On higher capacity nondismountable media, ready availability of an effective flaw table mechanism is essential.

Whenever a disk data read or write error occurs under RP/M, the number of the block containing the bad sector is stored in a fixed location of low memory at address 003B. When, for example, the message

```
Disk error on B: Bad sector
```

occurs, if you want to enter the number of the error producing block into the flaw table, press control-C to warmboot the system, then call the automatic flaw table processing program with the command

```
LOCKOUT B:
```

LOCKOUT retrieves the block number xxxx from location 003B and displays this number in the message

```
Block to be flawed:  xxxx  hex
Ok? (Y/N):
```

Answer "Y" to enter block number xxxx into the flaw table. After entering this block into the flaw table, or if you enter something other than "Y," LOCKOUT shifts into conversational mode allowing other block numbers, if desired, to be entered into the flaw table.

When you respond by entering "0" (zero) to the message

```
Block number in hex (or 0 if none):
```

LOCKOUT will display all of the block numbers that appear in the current flaw table, and then control will return to the system via warmboot.

Read Errors vs. Write Errors

The best time to "flaw the disk," by trapping a disk data error and entering the block into the flaw table, is when you have occurence of a disk write error. Once the block number is entered into the flaw table, you can write files without again encountering that particular block.

Handling read errors can be more troublesome. One way to treat read errors is to accept the bad record by pressing carriage return, for example, instead of control-C. You would probably do this if there is some possibility that you can salvage the remainder of the information in the file by completing the copy operation. When the file copy is complete, call LOCKOUT, flaw the block, and then erase the file that created the read error.

When you copy from one file to another on the same disk, depending on your CBIOS, you may not be able to determine whether a disk data error as reported by the RP/M message

```
Disk error on d: Bad sector
```

is a read error or a write error. For an effective error treatment, in either case, reject the error condition with control-C, then flaw the disk with LOCKOUT.

If the error was a write error you probably will not again encounter that block; but if it was a read error, you certainly will encounter that same block each time you attempt to read that same file. If the block number appears in the flaw table, simply erase the file that created the read error and then do a warmboot. When a block number appears in the flaw table and does not appear in any other active directory entry, you will not encounter that block during any read or write operation. There is, incidently, no harm created by allowing a block number to appear more than once in the flaw table, although once is enough.

Erasing the Flaw Table

To remove the flaw table from a directory you must first clear the "read-only" file attribute. This can be done with the command

```
STAT d:DISKFLAW.TBL $R/W
```

The flaw table can then be cancelled with the command

```
ERA d:DISKFLAW.TBL
```

where d is the name of the drive.

Summary

FLAWTBL can be used to test all blocks in the data space of a disk and will reserve any unuseable blocks in a disk flaw table, as RBT entries in the directory of a file named DISKFLAW.TBL, a read-only system file. The read-only attribute deflects any attempt to write into this file; while a zero record count deflects any attempted read. Under standard system operations, this file can be neither read nor written, the sole purpose for its existence being to isolate unuseable blocks.

If a disk data error occurs while the disk is being used by an applications program, the block involved can be entered into the flaw table by LOCKOUT.